Data Postdiction for Distributed Systems

Trevor Petersen
*Dept. of Computer Science, University of Pittsburgh, Pittsburgh, USA
‡ttp10@pitt.edu

I. Introduction

The need for efficient compression and retrieval speeds has become increasingly important in facilitating faster computational analysis. The cost of storage is decreasing at a much slower rate compared to the exponential growth of computational power. Even with the increased computational power, the large volume of data being collected prevents all of the data from being considered as part of the current analysis workflows. Data Postdiction [1], [2] is a new way to implement data decay, making a statement about the past value of some tuple, which is removed to free up space. Data postdiction relies on machine learning (ML) algorithms to abstract data into compact models that can be stored and queried when necessary [2]–[5]. This allows for the deletion of entire data columns, with one or more columns being preserved to recover the decayed columns using ML.

Postdiction behaves similar to lossy compression, that is, postdiction exhibits a *storage-accuracy* trade off, however, data postdiction allows the amount of error to be bounded. A program argument e represents the error constraint and the postdiction pipeline ensures that every recovered value is with e % of its original. Additionally, in contrast to conventional compression used in practice, postdiction can retrieve individual data values without recovering the entire compressed dataset. This vast increase in speed is shown in Table I

Figures 1 and Figures 2 shows Data Postdiction is capable of outperforming four modern lossless algorithms. In these two datasets (which are detailed in section II-B), Postdiction outperforms the best compression algorithm with between .5% and 2.5% error. Thus this method can effectively reduces database sizes and achieve significant compression levels. Still data postdiction is a slow process that involves clustering and training models on very large amounts of data and memory. This paper explores some ways in which we can alleviate these issues by leveraging distributed systems to allocate work across various nodes.

Here are the contributions of this paper:

- The paper explores and quantifies issues related speed and performance of the data postdiction pipeline.
- The paper proposes a two phase approach to data postdiction, with the existing training phase being followed by fitting of new data.
- The paper details and compares three different methods of fitting new data.
- The paper explores what federated learning may look like for postdiction.

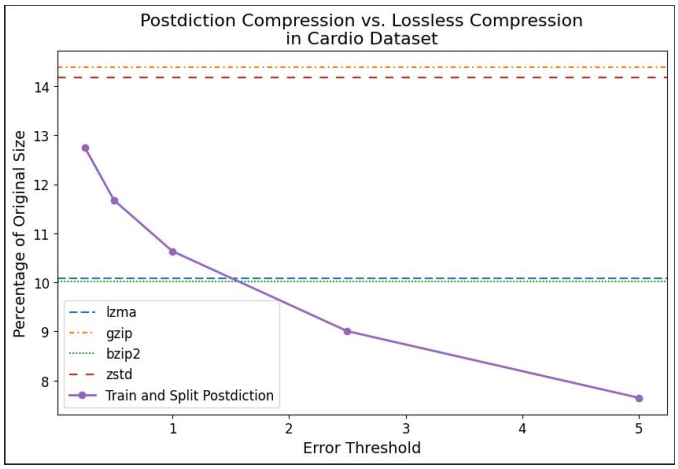

Fig. 1. Postdiction Reduction vs. Lossless Compression in Cardio Health Dataset

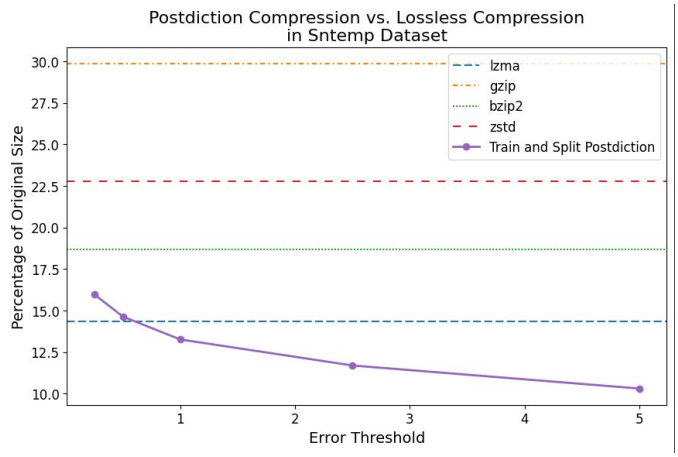

Fig. 2. Postdiction Reduction vs. Lossless Compression in River Dataset

II. Project Overview

This section will give an overview of a few aspects of the project, including information about the problem, datasets used through out the project,

A. Problem Overview

The previous iteration of postdiction incurred a prohibitively long training time and also was incapable of training and clustering on data that did not fit into memory. Table I shows how data postdiction compares across a few different error thresholds compared to a three lossless compression

TABLE I
THE RESULTS OF COMPARING LOSSLESS COMPRESSION AND THE POSTDICTION PIPELINE USING LR MODELS

Method	Error Threshold	Average Time to Reduce Data (ms)	Average Lookup for Single Row (ms)
LZMA	0%	350616	12774
GZIP	0%	75920	2284
BZIP2	0%	19535	12331
LR	0.25%	3351718	0.000612
LR	1%	994572	0.000612
LR	5%	601806	0.000612

algorithms. The table spells out the stark difference in time to reduce for the *sntemp* dataset. This is especially concerning for a compression scheme, in which large data files are the target.

B. Datasets Used in Experimentation

The postdiction pipeline was evaluated on following two datasets:

- **Cardio Healthcare Dataset** (Cardio): This smaller, anonymized healthcare dataset includes 70,000 records (about 2,941 KB) and was used in prior research [6]. The analysis focused on five numerical columns, while the remaining columns-simple binary categorical values (e.g., a "1" or "0" indicating whether a patient smokes) were excluded from consideration.
- **River Sensor Dataset** (Sntemp): This larger dataset contains 830,088 records (about 177.182 MB) from river sensors (e.g., waterflow, rainfall, humidity, air pressure) in the Delaware River Basin [7]. The analysis included 21 numerical columns and excluded the index and date columns, as these data types are less suited for postdiction and would require alternative compression techniques.

C. Relevance to Distributed Systems

While the postdiction project was not originally designed for distributed systems, it can significantly benefit its performance if it employs these systems correctly. Section III will go into depth about training and fitting phases as well as the three newly proposed fitting methods. One important aspect about the fitting methods that makes it different from training, is that it can parallelized and therefore performed across different nodes.

In an IoT setting, a central node with significant computational resources could perform this more intensive training, and share all of the models it trains with each device so it can perform fitting on the edge. This would reduce the size of the data even before it sends it to the central node (or other location) for storage.

D. Note

At the time of the projects conception the pipeline was using kmeans as the primary clustering algorithm. Since then, a new approach has been employed that clusters data purely off its position relative to the last model trained. This *Train and Split* saw an up to **10x** increase in speed. This shows that the biggest hindrance in training these models was the clustering

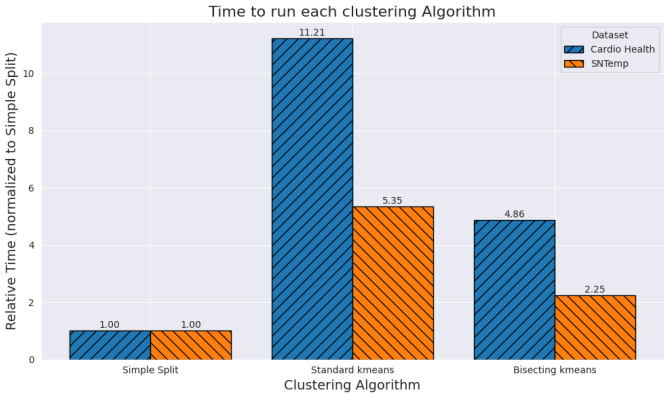

Fig. 3. Comparison of training time for Train and Split (Simple Split), kmeans, and bisecting kmeans clustering

on these large datasets. When performing quantitative analysis on the speed increase of the including the fitting phase, I included both postdiction with kmeans (to understand the original motivation) and the Train and split (to understand how it stacks up against the current best option). Figure 3 displays the difference in these runtimes. It is normalized to the time of the Train and Split approach for both datasets to account for the large difference in training times due to the size difference of the data.

III. THREE METHODS OF FITTING DATA

This section discusses the two-phase approach to postdiction and the three fitting methods devised to quickly place new data values into existing models quickly. Exhaustively testing a point on many models is too slow so these methods are proposed instead.

A. Training and Fitting Phases

The new postdiction approach is divided into two primary phases: the *training phase* and the *fitting phase*. During the training phase, a representative data sample is selected, on which clustering, model training. In the fitting phase, new data (in batches) is assigned to clusters based on these pre-trained models. For this reason, the paper will refer to these approaches as the batching approaches. This two-phase approach can significantly improve speed, as the fitting phase processes data much faster than the training phase.

However, this approach involves a trade-off: if the sample used in training is not representative, the resulting models may fail to generalize, potentially leading to a higher rate

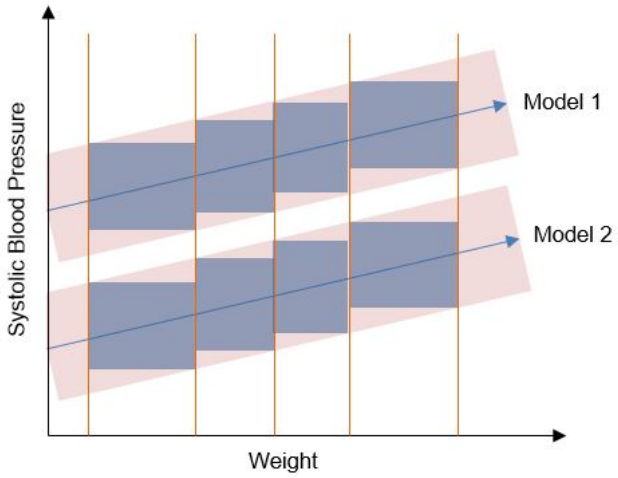

Fig. 4. Visual Representation of Data Fitting Methods 2 (Interval-Based Indexing) and 3 (Array-Based Index Optimization)

of outliers and reduced storage savings. A potential solution to this issue, which warrants further research, is to utilize continuous learning during the fitting phase for training new models on the outliers.

B. Model Sorting and Binary Search

In this method, models were sorted based on a specific plane (e.g, the y-axis or the median x-value of the data). To fit a new data point, an approximate location for its corresponding model was first estimated. If the point fell above or below the first model, the algorithm sequentially moved through the sorted models until either a match was found within the error threshold, or the point was determined to be an outlier (either outside all models or between two models). This method relies on two assumptions: (1) that the models are similar in nature—often the case with LR models that align with data correlations and tend to have comparable slopes—and (2) that the x-value approximation effectively predicts the behavior of the model, which holds true when the model's slope is near zero. While this method offers speed improvements, it remains limited by the inefficiency incurred by running models on individual inputs, which is a slow way to test many values.

C. Interval-Based Indexing (Tree Implementation)

To further speed up the fitting phase, an interval-based index structure dividing the x-axis into intervals was introduced. Each interval on the x-axis begins where the previous one ends, forming a continuous set of intervals covering the range of the input. For each interval, the maximum and minimum y-values for each model are identified. A secondary interval tree is then constructed for efficient traversal and matching. By using this tree, the correct model can be identified without ever needing to run it. If a point matches an interval in this tree, it can be immediately assigned to the model without explicit verification.

This method also involves certain assumptions and trade-offs. Specifically, it assumes that each model's minimum and maximum values lie at the interval edges—a property that is a characteristic of LR models but potentially less applicable to other model types. To avoid false positives (i.e., assigning points to a cluster when this would actually violate the error threshold), the y-interval boundaries are constrained, as shown in Figure 4. However, this adjustment may lead to some false negatives (i.e., assigning points as outliers when they are within the model's error threshold). This issue, while more likely with models with steeper slopes, can be mitigated by using smaller x-intervals in denser data regions where many new points are expected to fit. Thus this method can provide a significant speed-up over the first approach, with false negatives occurring infrequently. To eliminate false positives entirely, the first method could be applied to all outliers to ensure they do not fit into any cluster.

D. Interval-Based Indexing (Array Implementation)

This improved method replaces x-interval trees and y-interval trees with a streamlined data structure based on arrays. In particular, two arrays are utilized: **a minimum value array** and a **maximum value array**, where each index represents an interval. For any value v to belong to an interval at index i, it satisfies the condition:

$$\text{minimum\_array}[i] \leq v \leq \text{maximum\_array}[i].$$

This approach consolidates the set of intervals into a contiguous memory block that can be scanned efficiently. By storing the arrays within a single memory page, scanning is significantly faster compared to traversing interval trees, as it leverages better cache locality. Consequently, fewer memory pages are accessed, and these pages are more likely to remain cached, reducing access times.

IV. Experimental Results

A. Batching in the Postdiction Pipeline on the Cardio Health Dataset

Figure 5 shows the difference in runtime for the 3 indexing methods. Additionally, it has two baseline measurements with no batching, one that uses the simple splitting approach and one that uses kmeans clustering (explained in Section II-D). It is clear that all of the batching methods offer a significant improvement from the previously used no batching approach with kmeans clustering.

While it may seem like like even the best method did not offer an speed up compared to the new Train and Split approach to clustering, there are two reasons why this is not necessarily true. First, the file size of the Cardio Train data is too small to significantly test the efficiency of the fitting methods in the three batching approaches. Normally, the first few batches that are fit are a bit slower than the ones that follow, as the memory and cache adjust to the workload. Secondly, the fitting methods can run concurrently across many different nodes or CPU cores. The set up now only utilizes one core and one thread. So this can be significantly

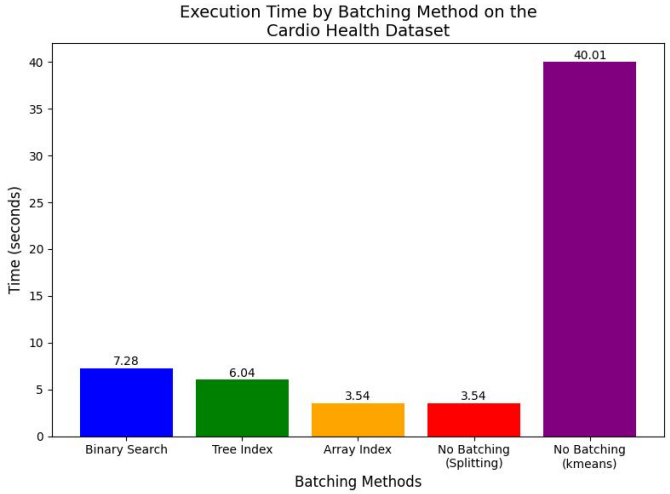

Fig. 5. A comparison of the runtime for the various batching methods on the cardio health dataset

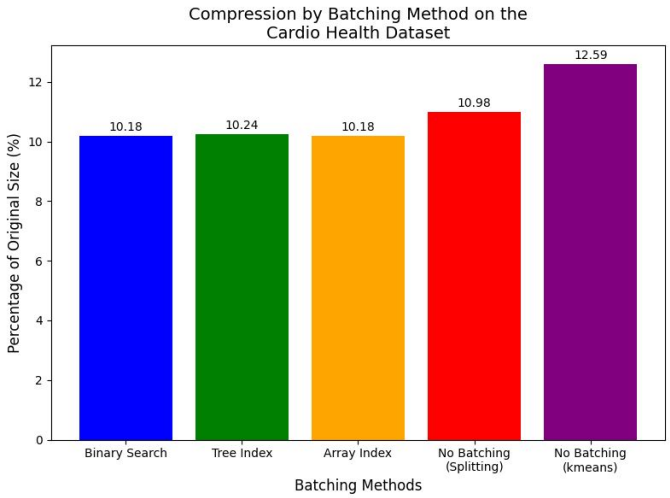

Fig. 6. A comparison of the compression for the various batching methods on the cardio health dataset

sped up by distributing the fitting workload (potentially even to IoT devices).

Figure 6 shows that there is little difference in the compression among all of the methods. Something interesting about this section is that the batching methods seem to have better compression rates than the no batching methods. The reason why the kmeans approach is so poor is that it will perform vertical splits (which result in a left and right cluster). When subsequent models are trained they ususally have low slopes and thus overlap. This results in more cluster than necessary. The reason why the standard no batching with splitting performs worse is that it is also allocating too many clusters. Additionally, the sample used for training is very strong due to the fact that the data is randomly distributed.

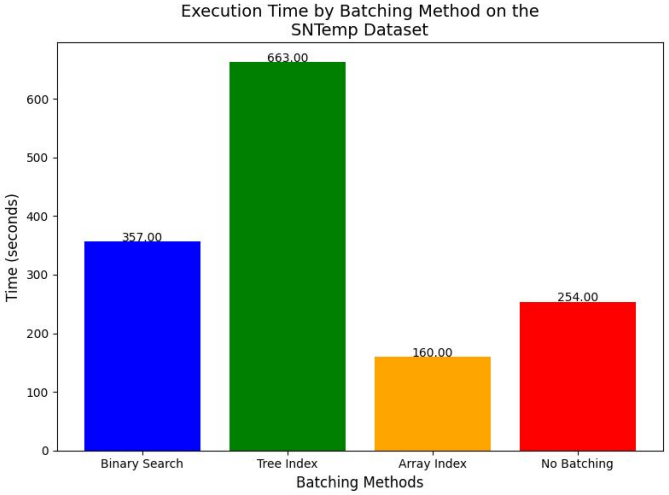

Fig. 7. A comparison of the runtime for the various batching methods on the sntemp dataset

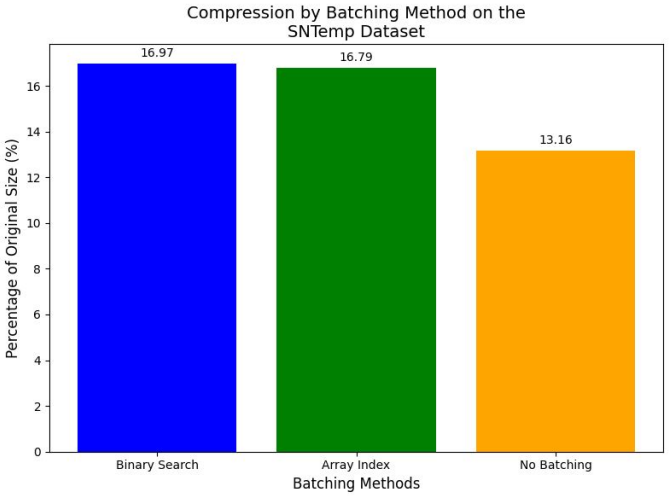

Fig. 8. A comparison of the compression for the various batching methods on the sntemp dataset

B. Batching in the Postdiction Pipeline on the SNTemp Dataset

In this section, there will no further analysis on the kmeans approach to batching. It takes too long to run on the SNTemp dataset. Please refer to Figure 3 to see an analysis of the relative run times difference between the kmeans and the training and splitting approach.

s

As shown in Figure 7 there is gains to be made with the batching method. This graph shows that the Array Index batching method was close to 40% faster than the traditional no batching approach. This is not even taking into account the potential gains to be made in the future by distributing the fitting workload across various cores or devices. The other two methods lag behind the simple split approach with no batching. Still they would significantly outperform the kmeans approach

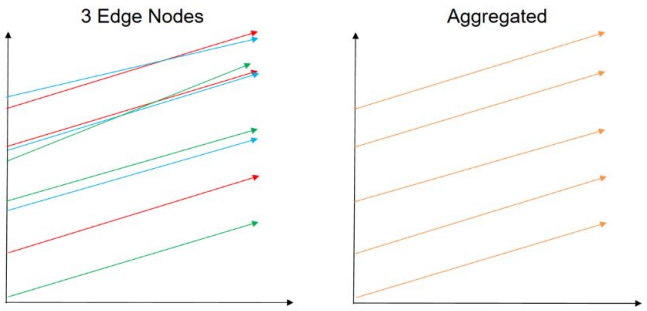

Fig. 9. Proposed Decayed Table Format

which was used at the start of the project. Another interesting thing to note here is the very poor performance from the tree based index. This was because the data here had many clusters and thus there were many jumps throughout the index trees which significantly hurt its speed.

Figure 8 shows that the best method for achieving the best compression is to use all of the data for training (perform no fitting). The Tree based interval index structure was not included here because it is logically the same as the array based interval index. These results are more significant than the previous section because the SNTemp dataset is larger and has more realistic variation in data, making it more difficult to sample. Future research could include how to do continuous training to alleviate the longer training times while not sacrificing compression.

V. Federated Learning

This section will quickly talk about what federated learning may look like in the postdiction setting. Firstly, federated learning could be valuable in the training phase of postdiction to include more data in training our models while not significantly hampering our run time performance.

Previously, I've considered using a learning method in which the central node aggregates models trained on the edge and tries has to also assign the points from these edge models into the new models. This approach would have to have a pretty contrived solution that uses lower error thresholds on the edge nodes and cluster each model along the x-axis to ensure every point from the edge model would fit into the new model. I think a more realistic and effective approach would be the one shown in Figure 9. This approach would use edge nodes to train models, combine the similar models, and produce a set of distinct models that covers all the data shards considered by every node. This would still require subsequent refitting of all the points but would help with reducing the training time and under sampling problems.

VI. Conclusion

In conclusion, this paper demonstrates the potential of data postdiction as a powerful alternative to traditional compression methods. By quantifying the speed and performance limitations of the current postdiction pipeline, I sought to enhance its efficiency through a two-phase approach that separates training and data fitting. This approach is designed to work in distributed systems (i.e. a network of IoT devices) warranting its relevance to this course. Additionally, the exploration federated learning for postdiction presents a scalable solution to the computational and memory demands of clustering and model training on large datasets. The results underscore the potential of postdiction to significantly reduce database sizes while maintaining low error thresholds and fast runtimes.

References

[1] C. Costa, A. Charalampous, A. Konstantinidis, D. Zeinalipour-Yazti, and M. F. Mokbel, "Decaying telco big data with data postdiction," in *19th IEEE International Conference on Mobile Data Management, MDM 2018, Aalborg, Denmark, June 25-28, 2018*. IEEE Computer Society, 2018, pp. 106–115.

[2] C. Costa, A. Konstantinidis, A. Charalampous, D. Zeinalipour-Yazti, and M. F. Mokbel, "Continuous decaying of telco big data with data postdiction," *GeoInformatica*, vol. 23, no. 4, pp. 533–557, 2019.

[3] A. Baskin, S. Heyman, B. T. Nixon, C. Costa, and P. K. Chrysanthis, "Remembering the forgotten: Clustering, outlier detection, and accuracy tuning in a postdiction pipeline," in *New Trends in Database and Information Systems - ADBIS 2023, Barcelona, Spain, September 4-7, 2023, Proceedings*, vol. 1850. Springer, 2023, pp. 46–55.

[4] G. Cormode, M. N. Garofalakis, P. J. Haas, and C. Jermaine, "Synopses for massive data: Samples, histograms, wavelets, sketches," *Found. Trends Databases*, vol. 4, no. 1-3, pp. 1–294, 2012.

[5] C. Costa, A. Charalampous, A. Konstantinidis, D. Zeinalipour-Yazti, and M. F. Mokbel, "TBD-DP: telco big data visual analytics with data postdiction," in *19th IEEE International Conference on Mobile Data Management, MDM 2018, Aalborg, Denmark, June 25-28, 2018*. IEEE Computer Society, 2018, pp. 280–281.

[6] S. Ulianova, "Cardiovascular disease dataset," 2019. [Online]. Available: https://www.kaggle.com/datasets/sulianova/cardiovascular-disease-dataset

[7] J. Zwart, K. S. Oliver, D. W. Watkins, M. J. Sadler, P. A. Appling, R. H. Corson-Dosch, X. Jia, V. Kumar, and S. J. Read, "Near-term forecasts of stream temperature using process-guided deep learning and data assimilation," August 2021.